BProVe: A formal verification framework for business process models

Corradini, Flavio; Fornari, Fabrizio; Polini, Andrea; Re, Barbara; Tiezzi, Francesco; Vandin, Andrea

Published in:
Proceedings of 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)

Link to article, DOI:
10.1109/ASE.2017.8115635

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

BProVe: A Formal Verification Framework for Business Process Models

Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, Francesco Tiezzi
School of Science and Technology
University of Camerino
Camerino, Italy
firstname.lastname@unicam.it

Andrea Vandin
DTU Compute
Technical University of Denmark
Lyngby, Denmark
anvan@dtu.dk

***Abstract*—Business Process Modelling has acquired increasing relevance in software development. Available notations, such as BPMN, permit to describe activities of complex organisations. On the one hand, this shortens the communication gap between domain experts and IT specialists. On the other hand, this permits to clarify the characteristics of software systems introduced to provide automatic support for such activities. Nevertheless, the lack of formal semantics hinders the automatic verification of relevant properties.**

This paper presents a novel verification framework for BPMN 2.0, called BProVe. It is based on an operational semantics, implemented using MAUDE, devised to make the verification general and effective. A complete tool chain, based on the Eclipse modelling environment, allows for rigorous modelling and analysis of Business Processes. The approach has been validated using more than one thousand models available on a publicly accessible repository. Besides showing the performance of BProVe, this validation demonstrates its practical benefits in identifying correctness issues in real models.

***Index Terms*—Business Processes, BPMN, Structural Operational Semantics, MAUDE, Software Verification.**

I. Introduction

A Business Process (BP) model generally describes a set of activities that an organisation should perform to fulfil a specific business goal [1]. In addition it is possible to use such kinds of models, in particular the so-called Collaborations, in order to describe the coordination of different organisations that cooperate to achieve a shared goal.

A BP model results from the synthesis of perspectives related to different aspects of an organisation, and on the activities that should be performed [2]. Among these, the following kinds of information are particularly relevant in order to derive an effective BP model. The *functional perspective* describes those units of work, also referred as activities, included in a BP that may be needed to reach a particular objective. The *behavioral perspective* specifies the behavior of the BP defining the control flow relationships between the included activities. The *organisation perspective* describes the different actors that are involved in a BP and their message exchange.

Such modelling approach has not specifically emerged in relation to the development of software systems. Instead, it has initially acquired consensus as an effective way to reflect on how an organisation operates, also in relation to possible collaborations with others. On the other hand, such kind of modelling activity constitutes an important thrust toward the introduction of software systems. As it happens for any modelling activity, the usage of a modelling notation to describe the reality of interest permits to reduce the communication gap between the different users of the model, so to keep the focus just on the relevant aspects. Interestingly, possible different users of BP models are domain experts and IT professionals. Therefore in the last years BP modelling has emerged as an effective approach in relation to requirements engineering activities [3], [4], [5], [6], in particular for the development of those software systems supporting the BP execution.

The relevance of BP modelling for software development is probably even more evident if we consider the notations inspired by BP modelling that have been proposed, with different levels of success, as executable languages. This is for instance the case of service orchestrations expressed using the Web Services Business Process Execution Language [7]. Another common use of BP languages in driving complex software systems concerns Process-Aware Information Systems, which dynamically coordinate different resources (people, applications, and information sources) according to what is defined in BP models. In summary, BP modelling has evolved to become an effective way to specify software characteristics, and even to program software systems. As a consequence, the relevance of formal verification for BP models has become more and more evident.

In the last years the Business Process Modeling Notation (BPMN 2.0) [8], an Object Management Group (OMG) standard, emerged as one of the most accepted proposals to define BP models. BPMN success comes from its versatility and capability to represent BP for different purposes. The notation acquired, at first, acceptance within business analysts and operators. Successively, it has been more and more adopted by IT specialists to lead the development and settlement of IT systems supporting the execution of a specified process model. This shift in the usage of the notation is particularly relevant and poses the basis for our work. Indeed, OMG does not provide a precise definition for the semantics of the notation. The lack of a precise semantics may not represent a big issue when the notation is used just for communication purposes. Instead its adoption for shaping IT systems, and even more importantly to apply model driven approaches to

978-1-5386-2684-9/17/$31.00 © 2017 IEEE

ASE 2017, Urbana-Champaign, IL, USA
Technical Research

automatic code generation, does require the definition of a precise semantics. To overcome such semantic gap, many different proposals can be found in the literature. Use of encodings and composition rules permitting to derive a Petri Net inspired model is a quite popular approach to provide the semantics of a BP (see, e.g., the seminal work in [9]). In this case, the analysis will be conducted on the derived Petri Net. Other approaches provide, instead, denotations of BPMN elements with terms in a given process algebra (see, e.g., [10], [11], [12]).

In this paper we provide an alternative verification framework that is based on the definition of a native semantics according to a Structural Operational Semantics (SOS) style [13]. The framework, as detailed in the following, has some distinctive characteristics:

- It does not make any assumption on the structural properties of the original model. Indeed, in order to make the definition of semantic rules easier, some approaches assume that models are defined according to good modelling practices. Well-structuredness, which informally asks for the usage of nested structures, is probably the most common assumption. On the other hand, it is quite common in practice that BP designers, in particular less experienced ones, do not generally follow such a recommendation. On the other hand some authors suggest that modellers should not be constrained by structural characteristics, so to freely represent the reality [14]. Thus it is not uncommon to find models with an *arbitrary topology*.
- It permits easy handling of BPMN elements that can have non-local effects, and that are difficult to handle with other approaches.
- It considers also BPMN communication elements, thus dealing with BPMN *Collaborations*. Hence, it enables the checking of issues possibly related to messages exchange.
- It makes easier to re-conduct the results of verification activities to the elements of the original model.

The semantics has been concretely implemented using the formal framework MAUDE [15]. This implementation has been integrated with the MAUDE model checker, to allow the automatic verification of properties of BPMN collaborations. The resulting verification component has been wrapped as a publicly available Web service, and a specific plug-in for the Eclipse modelling environment has also been developed. In particular, the resulting tool allows the user to select the collaboration properties he wants to verify out of a predefined list of intuitive behavioral properties expressed in natural language, and then to automatically generate logical formulas and to model check them over the BPMN model. This makes transparent to the user the use of formal methods, by offering a modern integrated development environment, equipped with verification facilities for BPMN models. The result is a complete tool chain for BP modelling and verification, which we called BProVe. The verification strategy has been also extensively validated using models from an open repository (http://bpmai.org/) [16], in order to derive its performance characteristics in relation to BP properties (i.e., soundness and safeness) relevant in software implementations. Notably, we report on a large set of publicly available models that do not satisfy such properties, witnessing the need of enriching BPMN modelling with rigorous analysis techniques.

Summing up, the major contributions of this paper are:

1) an *efficient implementation* of a formal operational semantics for BPMN within the MAUDE environment, which enables the *formal analysis* of BPMN models;
2) a complete *tool chain* presenting a modelling environment and a service for the automatic verification of properties over the designed BPMN models;
3) an extensive *validation* of the approach.

The rest of the paper is organised as follows. Section II provides an overview of BPMN. Section III discusses the main features of the verification framework we propose, focusing on its implementation and on the supported verification strategies. Section IV provides an overview of the tool chain. Section V illustrates the results of the validation experiments. Section VI reviews related works mainly focusing on existing tools. Finally, Section VII concludes by also touching upon directions for future work.

II. Background Notions

Having understood the rationale of considering Business Process management on software development for IT systems, in this section we introduce some basic notions. In particular, we first discuss the phases of the BP life-cycle, and we introduce BPMN as the reference language for modelling BP. Then, we present relevant properties to be checked.

There are many proposals for the generic life-cycle of BPs, deployed within IT infrastructures; here we refer to a slightly revised version of the one proposed in [17]. It consists of four phases: (i) *Design and Modelling*, where domain requirements are collected to produce a model suitable to represent as-is or to-be scenarios in organisations; (ii) *Analysis*, where syntactic, structural and behavioral issues of the model are detected; (iii) *Enactment and Execution*, where the model is deployed based on the underlying IT infrastructure; (iv) *Monitoring and Improvement*, where functional traces and non-functional measures from the process execution are collected in order to identify bottlenecks. In this paper we mainly concentrate on the *Design and Modelling* and *Analysis* phases, which are usually completed in an iterative method till reaching a stable release of the model.

Focusing on *Design and Modelling*, several languages and graphical notations have been proposed to represent process models with differences in the level of formality. BPMN, which has been standardised by OMG [8], is currently acquiring a clear predominance thanks to: (i) its intuitive and graphical notation that is accepted by industry and academia; and (ii) the support provided by a wide spectrum of modelling tools (currently more than 50, see http://www.bpmn.org for a detailed list). Here we discuss the BPMN elements supported by BProVe (see Fig. 1). **Pools** represent participants or or-

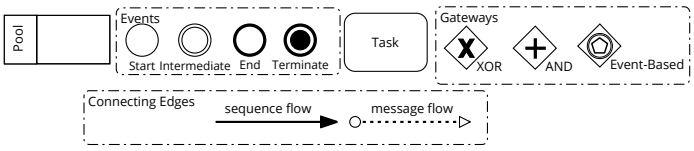


Fig. 1. BPMN Notation (an excerpt).

ganizations involved in the collaboration, and provide details on internal process specifications and related elements. Pools are drawn as rectangles. **Tasks** represent specific jobs to be performed within a process. Tasks are drawn as rectangles with rounded corners. **Gateways** manage the flow of a process both for parallel activities and choices. Gateways are drawn as diamonds and act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). Different types of gateways are available: an *XOR gateway* describes choices, an *AND gateway* enables parallel execution flows, and an *Event-Based gateway* activates its outgoing branches according to the taking place of catching events. **Events** are used to represent something that can happen. An event can be a *Start Event*, representing the point from which the process starts, an *Intermediate Event* representing something that happens during process execution, or an *End Event* representing the process termination. Events are drawn as circles. We also refer to a particular type of end event, the *Terminate End Event*, displayed by a thick circle with a darkened circle inside; it stops and aborts the running process. **Connecting Edges** connect process elements in the same or different pools. *Sequence Edge* is a solid connector used to specify the internal flow of the process, thus ordering elements in the same pool, while *Message Edge* is a dashed connector used to visualise communication flows between organisations.

Notably, we consider a subset of BPMN elements regularly used to design process models in practice. Indeed, we have selected such subset of BPMN elements by following a pragmatic approach and only retained the features actually used in practice (corresponding to less than 20% of its vocabulary [18]). Therefore, even if we focus on a restricted number of elements, we do not consider such design choice a major limitation of the work. Also, we remark that, if necessary, we might extend our framework to cover further elements. The only element that could present some challenges is probably the *OR-join* gateway. For such an element the BPMN standard provides a quite articulated description in which the behavior depends on the possibility that an upstream token could reach or not one of the flows entering the gateway itself.

Given that we focus on properties for BP *analysis*, in this paper we stress the importance of formal verification to check behavioral correctness. In the experimental section we consider the most relevant properties from the point of view of implementing a software system based on a BP model. In particular, we refer to *soundness* and *safeness* as formally defined in [19], [9], and [2], and for which we provide consistent LTL characterisations in Section III-C. Informally, *soundness* can be described as the combination of three basic characteristics concerning the dynamic behavior of a process model: (i) *Option to Complete*, requiring that a process instance can always complete, once started; (ii) *Proper Completion*, requiring that there exists no running or enabled activity for this instance when the process instance completes; (iii) *No dead activities*, requiring that a process model does not contain any dead activity, i.e., for each activity there exists at least one producible trace which contains the activity. On the other hand, *safeness* refers to the occurrence of no more than one token at the same time along the same sequence edge of a process instance. These properties naturally extend to process collaborations, requiring that the process instances of all involved organisations satisfy them.

The satisfaction of these properties is generally considered a minimal guarantee to avoid unexpected behavior of a BP model [17].

III. BProVe Framework

This section presents the proposed verification framework. First, we summarise the distinctive aspects focusing on BPMN modelling principles. Then, we exemplify the defined Structural Operational Semantics and its implementation in MAUDE. Finally, we discuss the properties we check.

A. Distinctive Aspects of the Proposed Framework

Our approach relies on a direct formalisation of BPMN semantics that, as a main distinctive aspect, supports models with an arbitrary topology. We are aware that this is not in line with good modelling practices and recommendations to use structuredness in modelling as a guideline to avoid errors [20]. However, in the real-world modelling practice most BPs designers do not follow such a guideline, as witnessed by a study we carried out on the BPM Academic Initiative repository. In fact, by following an arbitrary topology, designers are free to model the process according to the reality they feel, without needing to provide structured models [14]. In this way, the modelling activity results to be less complex [21] and more expressive [22], [23]. Therefore, we believe that considering models with an arbitrary topology will facilitate our verification framework in having a real impact on the development of process-aware systems, because all process designers are supported in their usual modelling activities, without imposing them any restriction or forcing them to follow a specific modelling style. Nevertheless, given that we consider a wider class of models, we can obviously verify those BPs that comply to the structuredness recommendation.

Another relevant contribution comes from the use of a direct formalisation of BPMN, avoiding typical problems given by encodings, where the semantics is not given in terms of features and constructs of the language, but in terms of low-level details of their translations. More specifically, in these approaches Petri Nets are often used as target language for mapping BPMN [9]. While for the basic BPMN modelling elements the encoding in Petri Nets is rather straightforward, for others such encodings are quite difficult to define. For example, the management of *termination end events*, permitting to abort a running process, is usually not supported. This is due

to the inherent complexity of managing non-local propagation of tokens in Petri Nets, which instead is natively supported by our semantics. Moreover, the main motivation to use Petri Nets is the availability of already developed tools supporting verification [24]. However, it is worth noticing that such tools work well for the basic Petri Nets formalism, but they are not anymore valid when considering extended versions of Petri Nets needed to support all the BPMN features, such as the management of task state evolution (e.g., enabling, running and complete) or different types of tasks. Our semantics provides an extensible framework that is able to potentially support all the features of BPMN, as Structural Operational Semantics permits to apply language extensions to cover any other BPMN feature without affecting the verification technique.

Another advantage of our framework, when compared to approaches based on encodings, is that it makes the verification of BPMN models more effective. This is because with the encodings the verification results refer to the low-level implementation of the models, and may be difficult to interpret them at BPMN level. On the other hand, our direct semantics enables formal reasoning on model properties at a level as close as possible to BPMN diagrams, so that diagnostic information can be directly reported on the diagram in a way that is understandable by process stakeholders. This is especially useful when many parties need to properly and quickly interact on the base of the models.

Concerning BP analysis, we provide a further novel contribution since we reason at collaboration level. This enables inter-organisational correctness which is still a challenge [25]. Thus, results of checking safeness and soundness with respect to BPMN collaborations differ from results obtained through encodings, which usually introduce a mapping at process level and then compose the processes in a collaboration by means of an inner transition [26]. In particular, differently from these approaches and in accordance with the BPMN standard, we do not impose any a-priori upper bound on the number of pending messages, which however has to be finite in order to perform the intended model checking analysis.

B. MAUDE Implementation of the Semantics

To practically enable verification of BPMN collaborations, we implemented within the MAUDE environment [15] the operational semantics for BPMN presented in [27][1]. MAUDE is an instantiation of rewriting logic [28] that has been used to specify the formal semantics of a wide variety of formalisms and languages [29], including C11 [30], [31], Java 1.4 [32], and JavaScript ES5 [33].

Since MAUDE specifications are executable, we obtained a formal interpreter for BPMN specifications. This enables formal verification of BPMN collaborations, e.g., by means of the MAUDE state space generator [15], or the MAUDE LTL model checker [34].

[1]Our MAUDE implementation of the BPMN semantics is available at http://pros.unicam.it/tools/bprove.

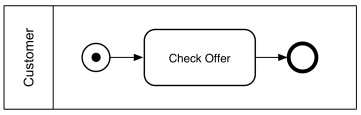


Fig. 2. Minimal Collaboration Example.

A detailed presentation would require to describe the semantics from [27], and rewriting logic, which is out of the scope of this paper. Rather, we summarise the main ideas used in our implementation by exemplifying the BPMN syntax and semantics that we implemented in MAUDE. We do this using a simple example consisting of a pool containing a start event, a task, and an end event. The example is depicted in Fig. 2, while its MAUDE encoding is provided in Listing 1.

```
collaboration(
 pool( "Customer" ,
  proc(
   start( enabled , "e1" . 0 )|
   task( disabled , "e1" . 0 , "o1" . 0, "Check Offer")|
   end( "o1" . 0 )
  ) , in: emptyMsgSet , out: emptyMsgSet ) ).
```

Listing 1. The MAUDE encoding of the collaboration in Fig. 2

As we can see from Listing 1, a collaboration is specified using the operator `collaboration`, which takes a set of pools as arguments (only one in Listing 1). A pool, defined by means of the operator `pool`, takes as argument its name, a BPMN process, and a set of incoming and outgoing messages to communicate with other pools (both empty in Listing 1). A BPMN process is specified using the operator `proc`, having as arguments the set of BPMN elements (separated by `|`) that compose it. The control flow is specified by the presence of tokens in the process elements, which are allowed to act only when enabled by tokens. In the example, we see that the start element is `enabled`, meaning that it has a token in input (denoted by the dot within the start event in Fig. 2), and hence it is allowed to initiate the process. The topology of the process is defined by the edges specified as arguments of the process components. In the example, the start node (operator `start`) is connected via sequence edge `e1` to the input of the task (operator `task`), whose output is in turn connected to the end node (operator `end`) via sequence edge `o1`. The number `0` associated to the sequence edges specifies that the sequence edges do not have a token associated yet. In MAUDE, the semantics is specified in terms of rewriting rules which are exhaustively applied by pattern matching on each generated state, until no new state can be generated. A rewriting rule has the following form:

```
crl [Label] : Term-1 => Term-2 if Condition(s) .
```

The keyword `crl` stands for conditional rewriting rule, whose optional name is specified in the square brackets. The body of the rule, `Term-1 => Term-2`, specifies that if `Term-1` can be matched on part of a state, then a new state will be obtained by: (i) removing the matched part from the state, and (ii) adding `Term-2` to the remaining part of the state. In the example, one of such terms can be the entire collaboration, a pool, a process, or a BPMN element. The `if` defines a

guard that has to be satisfied by the considered state in order to enable the application of the rule. In case no condition is required, then the `if` clause is omitted, and the keyword `rl` is used rather than `crl`.

```
crl [SketchOfRuleForProcesses] :
    el1 | RestOfProcess => el2 | RestOfProcess
 if el1 => el2 .

crl [SketchOfRuleForPools] :
    pool("Name",proc(proc1)) => pool("Name",proc(proc2))
 if proc1 => proc2 .
```

Listing 2. Sketch of rules for part of the semantics of processes and pools.

The BPMN semantics from [27] is multi-layer, in the sense that it has rules for collaborations (layer 4) that depend on rules for pools (layer 3), which in turns depend on rules for processes (layer 2), triggered by rules for single BPMN elements (layer 1). Roughly, the semantics is given in this form: if a BPMN element `el1` can evolve in an element `el2`, then a process `proc1` containing `el1` can evolve in a process `proc2` containing `el2`, and similarly for the higher layers, if necessary keeping into account interactions with other processes or pools. This can be mimicked in MAUDE using the condition `if el1 => el2` shown in the abstract sketch of rules for processes and pools provided in Listing 2.

For easiness of presentation, we just report simplified versions of the rewriting rules that are applied in order to let the collaboration in Listing 1 evolve in the new collaboration (and hence in the new state) in Listing 3, where the start element consumed the input token propagating it to the sequence edge `e1`, and consequently changed status from enabled to disabled. The resulting state is depicted in Fig. 3.

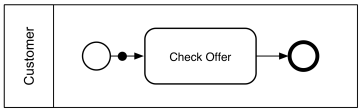


Fig. 3. Minimal collaboration example after one step of execution.

```
collaboration(
 pool( "Customer" ,
  proc( start( disabled , "e1" . 1 ) |
   task( disabled , "e1" . 1 , "o1" . 0, "Check Offer")|
   end( "o1" . 0 )
  ) , in: emptyMsgSet , out: emptyMsgSet ) ) .
```

Listing 3. A successor state of Listing 1, corresponding to Fig. 3.

First of all, we need a rule specifying the semantics of `start` elements, shown in Listing 4.

```
rl [E-Start] :
   start( enabled , IEName . IEToken )
=> {tUpd(IEName . IEToken)}
   start( disabled , IEName . IEToken + 1 ) .
```

Listing 4. Sketch of rule for task elements.

In the rule, `start(enabled, IEName . IEToken)` is the element on which the rule acts. It is a `start` element with status set to `enabled`. The symbols `IEName` and `IEToken` are variables that can be matched with any edge name and value of token, respectively. The rule establishes that the matched start element will change status to `disabled`, and will increase the counter of its outgoing sequence edge. The term `tUpd(IEName . IEToken)` is a label containing information on the executed action which will be used by the rules of the higher layers. Intuitively, in the example the label is used to propagate the update on the token to all occurrences of the sequence edge appearing in other BPMN elements. To give a hint on the strict relationship between the MAUDE implementation of the BPMN semantics and its formal presentation given in [27], just for this rewriting rule we report below the corresponding SOS rule:

$$(E\text{-}Start) \quad \bigodot \xrightarrow{e.n} \quad \xmapsto{+e.n} \quad \bigcirc \xrightarrow{e.n+1}$$

The one-to-one correspondence between the rewriting rule and the SOS one is clear, despite the different notation (which we do not explain here).

The rule for tasks triggers the execution of the rule for the processes sketched in Listing 5. Roughly, such rule propagates the changes from a single BPMN element to the entire process: (i) the label is associated to the entire process rather than to the single BPMN element; and (ii) the token counter in all copies of the interested sequence edge occurring in other BPMN elements is updated using the auxiliary function `markingUpdate`. We do not present in details the `markingUpdate`, as it just scans all BPMN elements within `RestOfProcess`. Note that this rule can be applied only in case the execution of the BPMN element creates a label `tUpd`. Other rules at the process level, not shown here, are provided in order to handle other kinds of labels.

```
crl [N-MarkingUpd] :
   ProcElem1 | RestOfProcess
=> {tUpd(Edges1)}
   ProcElem1' | markingUpdate(RestOfProcess,tUpd(Edges1))
if ProcElem1 => {tUpd(Edges1)}ProcElem1' .
```

Listing 5. Sketch of rule for processes that propagates token updates.

Listing 6 shows how the dynamics from lower layers propagate at pool level. Essentially, as discussed above, the rule in Listing 6 establishes that if the process `ProcElem1` of the pool can evolve in the new process `ProcElem1'` by creating label `Action1'`, then the entire pool evolves in a new pool where `ProcElem1` is substituted by `ProcElem1'`, generating a new label `collab(OrgName1, Action1')` (recording the organisation name) to be considered at the level of collaboration. The condition `internal(Action1')` imposes that this rule handles only actions internal to the pool, while actions that regard communications with other pools are handled by other rules, omitted here.

```
crl [C-Internal] :
   pool(OrgName1,proc({Action1}ProcElem1),
        in:inputMsgSet ,out:outputMsgSet)
=>{collab(OrgName1 , Action1')}
   pool(OrgName1,proc({Action1'}ProcElem1'),
        in:inputMsgSet ,out:outputMsgSet)
if ProcElem1 => {Action1'}ProcElem1' ∧ internal(Action1')
```

Listing 6. Sketch of rule for pools.

Finally, the rule for collaboration level shown in Listing 7 propagates the updates at the level of collaboration. As in the case of the layer of pools exemplified in Listing 6, this rule

focuses on action internal to the pool. Other rules are defined to handle interactions between pools.

```
crl [C-Interleaving] :
   collaboration( Pool1 | Coll2 )
=> collaboration( Pool1' | Coll2 )
if Pool1 => {CollAction1} Pool1'∧ internal(CollAction1)
```

Listing 7. Sketch of rule for collaborations.

C. Verification

Verifying properties of BPMN collaborations is useful to ensure their correct execution. Using our MAUDE implementation of BPMN we can verify those properties that can be expressed in terms of LTL formulas [35], successively checked using the MAUDE LTL model checker [34].

The formulas we show here are obtained as composition of the following basic cases:

- <> ϕ, where the operator <> (corresponding to the LTL operator F) is used to verify if a formula ϕ **eventually** holds. That is, in any possible execution path we always encounter a state where ϕ holds.
- [] ϕ, where the operator [] (corresponding to the LTL operator G) is used to verify if a formula ϕ **globally** holds. That is, ϕ holds in all states encountered in any possible execution path.
- ϕ -> φ, where the operator -> is the standard boolean implication.

In order to verify the properties mentioned above of a model, say `BPmodel`, we have to execute the following MAUDE command:

```
red modelCheck(BPmodel,φ) .
```

Listing 8. MAUDE command to run LTL model checking.

In our tool we focus on verifying the Soundness and Safeness properties of BPMN models. In the rest of the section we exemplify the formulas we used to study those properties.

As we know from Section II, the Soundness property can be encoded in terms of three simpler ones: *Option To Complete*, *Proper Completion* and *No Dead Activities*. Notably, as these properties refer to a single pool, they have to be checked for each pool in the collaboration. Anyway, such checks are carried out over the overall collaboration model, in order to take into account the message exchange among organisations.

Option To Complete. This property relies in turn on two properties: *aPoolCanStart* and *aPoolEnds*. The former checks if a token is present in a start element of a given pool. Instead, the latter checks if a token is present in an end element of a given pool, implying that the pool completed the execution. The property is encoded has shown in Listing 9.

```
[](aPoolCanStart(poolName) -> <>aPoolEnds(poolName))
```

Listing 9. Property Option to Complete.

The above formula verifies that from any state ([]) in which the pool can start (`aPoolCanStart(poolName)`), we eventually reach a state (<>) where the pool completes its execution (`aPoolEnds(poolName)`). Interestingly, this formula adheres to the well-known *response property pattern* (see, e.g., http://patterns.projects.cs.ksu.edu/documentation/patterns/response.shtml)

Proper Completion. This property checks that a pool always correctly completes its execution. In particular, we check that whenever a token reaches the end of the pool, then no other token remains unused within the pool. To verify this property we rely on two properties: *aPoolEnds* and *NoDanglingToken*. The latter checks that the pool does not contain other tokens. The property is encoded has shown in Listing 10.

```
[](aPoolEnds(poolName) -> NoDandlingToken(poolName)).
```

Listing 10. Property Proper Completion.

In words we verify that whenever a pool completes its execution, then no other token remains in the pool. Differently from Listing 9, now the right-hand side of the implication does not have a <> operator because we check the *NoDanglingToken* condition on the same state that satisfied *aPoolEnds*. ***No Dead Activities***. This property relies on the verification of the condition *aTaskRunning*, which establishes that a given task can be set, at least once, in the status Running (meaning that the task is currently being executed). If this property holds for all the tasks in the model, then the model has No Dead Activities. The property is encoded as shown in Listing 11.

```
<> aTaskRunning(taskName) .
```

Listing 11. Property No Dead Activities.

Safeness. This property can be encoded in terms of one single condition only, *safeState*, as shown in Listing 12.

```
[] safeState(poolName) .
```

Listing 12. Property Safeness.

As shown in Listing 13, *safeState* evaluates to true in states that satify the auxiliary function *noMultipleToken*, which verifies that on each sequence edge there is at most one token.

```
ceq collaboration(
 pool( Org1Name ,
  proc( {Action1}ProcElements1 ),
       in: inputMsgSet ,out: outputMsgSet ) |
 Coll1 ) |= safeness(Org1Name) = true
if noMultipleToken(ProcElements1) = true .
```

Listing 13. Safeness Implementation

IV. BProVe Tool Chain

To better clarify the contribution of our Tool Chain, we report in the Sequence Diagram of Fig. 4 a typical usage scenario, while the interested reader can find further information on the tool and its usage in [36]. The considered scenario shows all the interactions between a User and the tool chain main components.

The user, after designing a BPMN model with the Modelling Environment, requests a check on the designed model (see screenshot in Fig. 5). Then, the Modelling Environment sends a request to the BProVe WebService asking for a parsing of the BPMN model. The BProVe WebService evaluates the model, verifying that in the model are not present BPMN elements that cannot be handled by the considered BPMN operational semantics. Based on the evaluation result, the sequence of

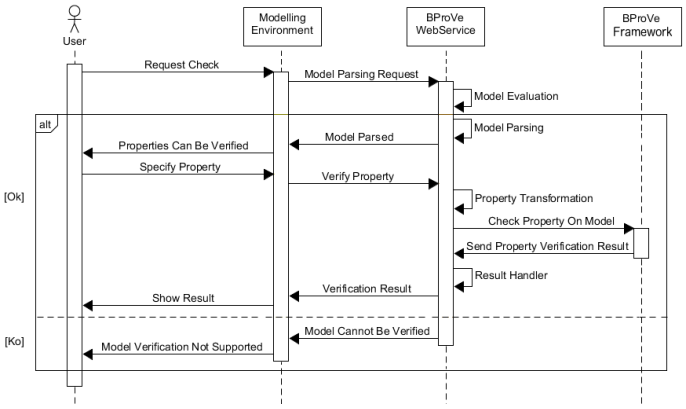


Fig. 4. BProVe Sequence Diagram.

interaction may differ. This is represented in the Sequence Diagram by an alternative block (delimited by the horizontal dashed line).

The part labeled "Ok" contains all the interactions that normally occur if the result of the Model Evaluation is positive, which means that the BPMN model contains only elements that are supported by our framework. In this case, the model is parsed and sent back to the Modelling Environment which asks the User to specify a property he/she wants to verify over the model. After the User specifies a property, the Modelling Environment sends a verification request to the BProVe WebService, which transforms the property in an LTL formula compatible with the MAUDE LTL model checker and it passes the property together with the models to the BProVe Framework. This latter component starts an instance of MAUDE loaded with the LTL MAUDE model checker and the MAUDE modules containing the semantic rules. Then, it verifies the property and sends the result back to the BProVe WebService, which properly formats the result and sends it to the Modelling Environment that will display it to the User. When the property verification result is negative, it means that the property is not verified over that BPMN model; if a counterexample is present, this is shown to the user. This information is visualised directly on the BPMN model (see the elements coloured in magenta in the screenshot shown in Fig. 5), thus facilitating the interpretation of the verification result to users, especially for those not familiar with the underlying formal verification technique. Instead, if the property verification result is positive, it means that the property is verified and a message stating this is displayed to the User.

The part labeled "Ko" contains the interactions that occur if the model presents elements that are not supported by our framework. In this case, the BProVe WebService informs the Modelling Environment on the model ineligibility for parsing, and, in turn, the Modelling Environment informs the User.

V. Validation

This section presents the experimentation we ran using our BProVe verification framework. The set-up and the results of the performed experiments are described below. The experiments can be replicated using a VirtualBox virtual machine containing an installation of our framework, available at http://pros.unicam.it/tools/bprove. Our validation has been shaped considering the following research questions:

- Are soundness and safeness already correctly handled by modellers, or instead modellers do release models violating such properties?
- Can BProVe actually and effectively support modellers in the verification of their models, specifically in relation to soundness and safeness properties?

A. Experimentation Set-Up

In order to validate our verification framework against real-word processes, we consider BPMN collaboration models provided by the BPM Academic Initiative (http://bpmai.org/) [16]. This is a collection including almost thirty thousand models codified using various process modelling languages. We chose it because it is particularly suited to investigate modelling practices thanks to its heterogeneity [37].

The raw dataset consists of 16 032 BPMN models, but we restricted to the latest revision of the models with 100% of connectedness[2]. A model without this level of connectedness includes disconnected fragments, which typically means that the model has not been finalized. Including such models in our validation would have resulted in verification data difficult to interpret. This gave us a dataset of 7 639 models with reasonable quality assurances. From these models we selected 1 245 models with more than 5 BPMN elements. This is because our focus is on collaboration, and 5 is the minimum number to have a pool exchanging a message with another. Considering our reference dataset, we perform a preliminary transformation step from .json (the repository format) to .bpmn (the format we manage), and then we check soundness and safeness. Verification has been carried out on the above mentioned Virtual Machine located into the GARR cloud platform (https://cloud.garr.it/). The machine runs Ubuntu 16.04.2 LTS 64 bits and it has 4 VCPU, and 8 GB of RAM.

B. Experimentation Results

To collect data we ran a massive analysis, checking if the models satisfy the properties discussed in Section III-C. From the 1 245 models, 1 026 (more than 82%) pass the parsing phase, meaning that they include elements we consider in our framework. This confirms that the selection of elements we did allows to deal with most practical cases.

The main analysis outcome is described in Table I. The table shows that only 55% of the considered models are sound. In particular, this is mainly due to the fact that Option To Complete and Proper Completion are satisfied by two different sets, each containing 57% of the models, while the percentage of models without dead activities is higher and accounts to 75%. In addition, we found that about 75% of the models is safe. These results are emblematic in proving the need of formal analysis techniques to verify the correctness of BPMN models.

[2]Connectedness evaluates the size of the largest connected sub-graph against the size of the overall model.

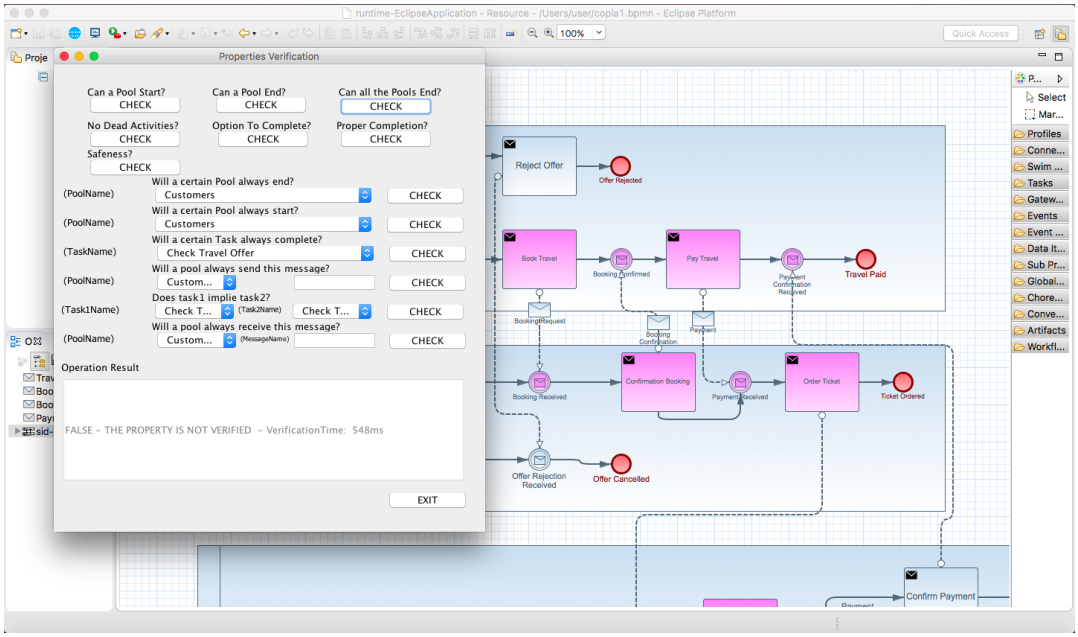


Fig. 5. BProVe User Interface.

TABLE I
FRACTION OF MODELS SATISFYING SOUNDNESS AND SAFENESS.

Property	*Models satisfying it*
Soundness	566 (55%)
- Option To Complete	584 (57%)
- Proper Completion	584 (57%)
- No Dead Activities	772 (75%)
Safeness	772 (75%)

Table II provides more insights on the complexity of the considered models, and on the time necessary to parse them. In particular, we classified the models in terms of the number of BPMN elements they contain (column *Class*). Column *AVG Elements* provides the average number of elements of models in the class, while column *Time* provides the average time in milliseconds necessary to parse each model in order to derive the format needed by MAUDE. As it can be observed, the parsing time slightly increases with the dimension of the model.

TABLE II
COMPLEXITY OF THE CONSIDERED MODELS AND PARSING TIME.

Class	*AVG Elements*	*Models*	*Time (ms)*
05–10	7	487	244
11–20	15	366	277
21–30	24	120	326
31–40	34	42	388
41–70	50	11	925

Table III provides more information on the complexity of the verification in terms of the time needed to check the considered properties. In particular, for each class of models the columns of the table report minimum, maximum, average, and median time as well as standard deviation. Time values are indicated as milliseconds needed to verify each property. It is worth mentioning that the values observed for class

TABLE III
EXPERIMENTAL RESULTS (IN MS.)

(a) Option to Complete.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	21 524	71	3	976
11–20	0	53 878	419	41	3 833
21–30	0	42 721	1 221	124	5 213
31–40	0	54 907	4 817	219	12 319
41–70	0	66 613	6 738	155	18 969

(b) Proper Completion.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	25 496	218	19	1 620
11–20	0	63 741	1 666	150	6 083
21–30	0	41 461	3 661	786	7 913
31–40	0	77 605	9 068	742	18 089
41–70	0	240 035	22 578	588	68 775

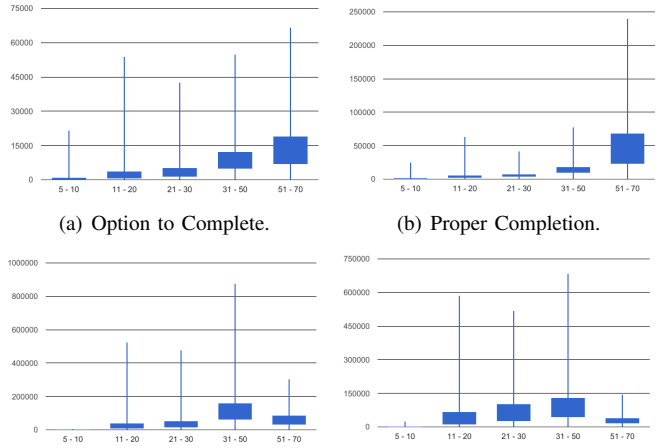
(c) No Dead Activities.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	5 802	61	2	295
11–20	0	523 742	5 473	225	39 921
21–30	0	477 232	13 647	1 554	52 296
31–40	0	877 553	59 765	1 860	159 123
41–70	0	303 632	29 555	606	86 744

(d) Safeness.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	25 569	207	5	1 364
11–20	0	587 756	9 913	127	68 316
21–30	0	519 712	25 752	937	103 418
31–40	0	685 198	42 505	1 157	130 045
41–70	0	144 513	15 256	890	40 985

41–70 are not fully significant given the small number of models belonging to such a class. Overall the observed data shows that properties can be verified in reasonable time and we were able to assess the properties on all the models in less than 15 minutes. We also report the trend related to the time needed to check soundness (Fig. 6(a)-6(c)), and safeness (Fig. 6(d)). In the diagrams the time required by the tool to verify properties is on the Y-Axis, while the class of the models is on the X-Axis. The diagrams provide a



(a) Option to Complete. (b) Proper Completion.

(c) No Dead Activities. (d) Safeness.

Fig. 6. Experimental Results

visual representation of some of the data reported in Table III. Overall, we can observe that, as it can be expected, times increase with the dimension of the models. There is also a high variability justified by the rather high values assumed by the standard deviations. This is not a surprise since, it is well-known that verification activities are particularly affected by the presence of notation elements leading to the interleaving of activities, such as parallel or pool statements, that are not always included in models. Nonetheless, the tool was able to provide an answer in reasonable time also for the most complex models in the repository. The analysis of the maximum values observed for each class somehow tell us that the maximum complexity of models, in terms of checking, is rather independent from dimensional characteristics. Indeed, as we also observed looking at the models associated to the maximum values, modellers tend to reach a sort of maximum complexity in the usage of elements leading to interleaving, and such complexity can be easily reached defining a model including 20 elements. Finally, considering the median values, we discover that the used repository includes, for a large fraction, very simple models. The value of the median is indeed much smaller (up to 20 times) than the average. This tells us that most of the considered models are rather simple and that few models present real issues for checking.

Summing up and answering the research questions posed at the beginning of this section, the usage of an open and widely used repository confirmed that it is not seldom to find models that violate relevant behavioral properties, also after their release. In addition the experiments show that our approach seems to be applicable in practice to realistic BP models.

VI. Related Work

Much effort has been devoted to the formalisation and verification of Business Processes [24], [38], [39]. Nevertheless, none of the cited works takes into account at the same time all the following key requirements: (i) enabling a direct formalisation being close to the BPMN standard, thus avoiding abstraction issues given by use of encodings; (ii) taking into account the collaboration aspects as key features of BPMN in large software development; (iii) providing a tool chain suitable to enable soundness and safeness verification on real scenarios.

In this section we refer to the most relevant tool-supported approaches available in the literature that inspired our work. We first consider the other direct formalisations enabling formal verification, and then we discuss those approaches supporting verification via encodings from BPMN to other well-known formalisms.

A. BPMN Direct Formalisations

With regard to direct formalisations, our contribution was mainly inspired by the one presented in [40]. The authors propose a BPMN formalisation based on in-place graph transformation rules; these rules are defined to be passed as input to the GrGen.NET tool, and are documented visually using BPMN syntax. With respect to our work, the used formalisation techniques are different, since we consider an operational semantics in terms of LTS, which allows us to apply verification techniques well-established for this underlying model, such as model checking. The definition of an operational semantics gives us the possibility to be tool interdependent rather than be constrained to tools specific for graph transformation rules. This is confirmed by the same authors that, for using their BPMN formalisation in the compliance verification between global and local process models, need a further transformation [41]. Focusing on verification the GrGen.NET tool aims to be used to verify workflow engines and service orchestration/choreography engines using BPMN [41]. The benefits of the solution are illustrated by means of a simple scenario that showcases the approach.

Other approaches are also proposed considering direct formalisations. El-Saber and Boronat proposed in [42] a formal characterisation of well-formed BPMN processes in terms of rewriting logic, using MAUDE as supporting tool. They discuss soundness of the well-formed BPMN models without introducing verification into practice, which is postponed as future work. This formalisation refers to a subset of the BPMN specification considering elements that are used regularly, such as flow nodes, data elements, connecting flow elements, artefacts, and swimlanes. Differently from our approach, it can be only applied to well-structured processes. Another direct formalisation is that proposed by Borger and Thalheim in [43]. They define an extensible semantical framework for BPMN using Abstract State Machines. It is based on the version 1.0 of BPMN, which does not include notation meta-model and gives more freedom to the authors in the interpretation of the language. Few years later, Kossak et al. [44] proposed a semantics based on Abstract State Machines for BPMN single process diagrams; differently from our proposal where collaboration aspects, such as pools and message exchange, play a key role, in [44] they are overlooked.

It is worth noticing that none of the approaches and supporting tools mentioned above has been properly validated, so

it is not clear which advantages and contributions they provide on real scenarios.

B. BPMN Formalisation via Encodings

The most common formalisations of BPMN are given via encodings to various formalisms, such as Petri Nets [9], [45], [46], [47], [48], or their extensions such as YAWL [49], [50] and ECATNets [51] [52], and process calculi [10], [53], [11], [54], [12], [55], [56], [57].

Regarding the encodings from BPMN to Petri Nets, the one proposed by Dijkman et al. in [9] is probably the most relevant contribution. The Petri Net resulting from the encoding of a BPMN model can serve as input to a Petri Net based verification tool for the static analysis of the model. The main contribution of the proposed tool is the transformation rather than the verification. Indeed the tool has been validated focusing on the transformation of only 13 models. The authors just state that they detected errors in some of the analysed models without giving details on effectiveness of the approach. Moreover, the approach proposed by Dijkman et al. is based on the version 1.1 of BPMN and, as the authors stated, it suffers from deficiencies that impact on the proposed formalisation. Moreover, differently from our approach, even if the encoding deals with messages, it does not properly consider multiple organisation scenarios.

Other relevant encodings are those from BPMN to YAWL, a language with a strictly defined execution semantics inspired by Petri Nets and able to support verification [19]. Among the proposed encodings, we would like to mention the ones by Ye and Song [49] and Dumas et al. [50]. The former is defined under the well-formedness assumption, which instead we do not rely on. Moreover, although messages are taken into account in the mapping, pools and lanes are not considered; thus it is not possible to identify who is the sender and who is the receiver in the communication. This results in the lack of capability to introduce verification at message level considering the involved organisations. The latter encoding, instead, formalises a very small portion of BPMN elements. In particular, limitations about pools and messages are similar to the previous approach: pools are treated as separate Business Process, while messages flow is not covered by the encoding. Focusing on supporting tools, the two solutions permit to transform BPMN to YAWL Nets enabling verification. Ye and Son implemented an open-source plug-in called BPMN2YAWL that uses ILog BPMN Modeler as a graphical editor to create BPMN models, and implements transformation and verification as ProM 5.0 plug-in [49]. As a proof of concept the tool has been tested using simple models. Decker et al. also consider BPMN as source language and YAWL as target language [50]. In this case, both the transformer and the modeller are Eclipse plugins, while the verification is supported by ProM 5.0. Again, as a proof of concept the tool has been tested to a limited number of models.

More recently, Kheldoun et al. [51], [52] proposed an encoding from BPMN to Recursive ECATNets, which can be expressed in term of conditional rewriting logic and given in input to the MAUDE LTL model checker. Even if we use the same model checker, the approach in [51] suffers from the encoding problems discussed above and, in particular, it does not consider messages in the encoding as well as the event-based gateway. Moreover, the authors illustrate the approach through three simple examples only, without extensively validating it.

Process calculi have been also considered as means for formalising BPMN. Among the others, Wong and Gibbons presented in [53], [10] a translation from a subset of BPMN process diagrams, under the assumption of well-formedness, to a CSP-like language based on Z notation using Haskell. This enables the introduction of a formal verification to check properties based on the notion of messages, like consistency between BPMN diagrams with different levels of abstraction and compatibility between participants within a Business Process collaboration [58]. Benefits of the solution are illustrated by means of a simple scenario.

Even if our proposal differs from the above ones, as it relies on a direct semantics rather than on an encoding, it has drawn inspiration from those based on process calculi for the use of a compositional approach in the SOS style.

VII. Concluding Remarks and Future Works

Thanks to the wide adoption of BPMN, Business Process modelling has acquired increasing relevance in the development of software systems. This is mainly due to the capability of BPMN to fill the communication gap between domain experts and IT specialists. However, the lack of a formal semantics prevents its full adoption, making unavailable the automatic verification of relevant properties impacting on the behavior of Business Processes.

Several approaches are available in the literature providing a formalisation to the BPMN standard. In this paper we rely on a native formal semantics for BPMN models, avoiding typical problems due to the encoding in different formalisms, such as the low accessibility of the analysis results.

Regarding verification of Business Processes, although the research community provided many proposals, only few of them are tool supported. Most of such tools are prototypes resulting from research projects, developed just for demonstration purposes, and hence are rarely used in practice. Validations on real case studies or wide model sets are also not available. In addition some of the prototypes are not even anymore maintained. In this paper, we presented a framework provided with a mature and easy-to-use tool-support, which we validated against more than one thousand models available on a publicly accessible repository. The results of the validation are reported and discussed in detail. Validation proves practical benefits and effectiveness of the approach.

In the future, we plan to extend our framework to other BPMN relevant characteristics, such as data management, time constraints and resource allocation, so to enable also a quantitative analysis for BPMN models. Finally, we intend to evolve the prototype, now implemented in MAUDE, using a "classic" programming language, so to further improve its performance.

REFERENCES

[1] A. Lindsay, D. Downs, and K. Lunn, "Business processes attempts to find a definition," *Information and Software Technology*, vol. 45, no. 15, pp. 1015–1019, 2003.

[2] M. Reichert and B. Weber, "Business process compliance," in *Enabling Flexibility in Process-Aware Information Systems*. Springer, 2012, pp. 297–320.

[3] O. Pastor, "Model-Driven Development in Practice: From Requirements to Code," in *SOFSEM 2017: Theory and Practice of Computer Science*, ser. LNCS. Springer, 2017, vol. 10139, pp. 405–410.

[4] J. Li, R. Jeffery, K. H. Fung, L. Zhu, Q. Wang, H. Zhang, and X. Xu, "A Business Process-Driven Approach for Requirements Dependency Analysis," in *Business Process Management*, ser. LNCS. Springer, 2012, vol. 7481, pp. 200–215.

[5] A. M. de Vasconcelos, J. L. de la Vara, J. Sanchez, and O. Pastor, "Towards CMMI-compliant Business Process-Driven Requirements Engineering," in *Quality of Information and Communications Technology*. IEEE, 2012, pp. 193–198.

[6] A. Aldazabal, T. Baily, F. Nanclares, A. Sadovykh, C. Hein, and T. Ritter, "Automated model driven development processes," in *ECMDA workshop on Model Driven Tool and Process Integration*, 2008, pp. 361 – 375.

[7] OASIS WSBPEL TC, "Web Services Business Process Execution Language Version 2.0," OASIS, Tech. Rep., April 2007, available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[8] OMG, "Business Process Model and Notation (BPMN V 2.0)," 2011.

[9] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[10] P. Y. Wong and J. Gibbons, "Formalisations and applications of BPMN," *Science of Computer Programming*, vol. 76, no. 8, pp. 633–650, 2011.

[11] D. Prandi, P. Quaglia, and N. Zannone, "Formal Analysis of BPMN Via a Translation into COWS," in *Coordination Models and Languages*, ser. LNCS. Springer, 2008, vol. 5052, pp. 249–263.

[12] F. Puhlmann, "Soundness Verification of Business Processes Specified in the Pi-Calculus," in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, ser. LNCS. Springer, 2007, vol. 4803, pp. 6–23.

[13] G. D. Plotkin, "A structural approach to operational semantics," *J. Log. Algebr. Program.*, vol. 60, no. 61, pp. 17–139, 2004.

[14] A. Polyvyanyy and C. Bussler, "The structured phase of concurrency," in *Seminal Contributions to Information Systems Engineering*. Springer, 2013, pp. 257–263.

[15] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All about maude-a high-performance logical framework: how to specify, program and verify systems in rewriting logic*. Springer, 2007.

[16] M. Kunze, P. Berger, and M. Weske, "BPM Academic Initiative - Fostering Empirical Research," in *Demonstration Track of the 10th International Conference on Business Process Management*, vol. 940, 2012, pp. 1 – 5.

[17] M. Weske, *Business Process Management*. Springer, 2012.

[18] M. Muehlen and J. Recker, "How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation," in *Advanced Information Systems Engineering*, ser. LNCS. Springer, 2008, vol. 5074, pp. 465–479.

[19] M. T. Wynn, H. M. W. Verbeek, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Business process verification-finally a reality!" *Business Process Management Journal*, vol. 15, no. 1, pp. 74–92, 2009.

[20] R. Laue and J. Mendling, "The Impact of Structuredness on Error Probability of Process Models," in *Information Systems and e-Business Technologies*, ser. LNBIP. Springer, 2008, vol. 5, pp. 585–590.

[21] B. Kiepuszewski, A. H. M. ter Hofstede, and C. J. Bussler, "On structured workflow modelling," in *International Conference on Advanced Information Systems Engineering*, ser. LNCS, vol. 1789. Springer, 2000, pp. 431–445.

[22] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," *Information Systems*, vol. 37, no. 6, pp. 518–538, 2012.

[23] A. Polyvyanyy, L. Garcia-Banuelos, D. Fahland, and M. Weske, "Maximal Structuring of Acyclic Process Models," *The Computer Journal*, vol. 57, no. 1, pp. 12–35, 2014.

[24] S. Morimoto, "A Survey of Formal Verification for Business Process Modeling," in *International Conference on Computational Science*, ser. LNCS. Springer, 2008, vol. 5102, pp. 514–522.

[25] R. Breu, S. Dustdar, J. Eder, C. Huemer, G. Kappel, J. Kopke, P. Langer, J. Mangler, J. Mendling, G. Neumann, S. Rinderle-Ma, S. Schulte, S. Sobernig, and B. Weber, "Towards Living Inter-organizational Processes," in *15th Conference on Business Informatics*. IEEE, 2013, pp. 363–366.

[26] W. M. van der Aalst, "Structural characterizations of sound workflow nets," *Computing Science Reports*, vol. 96, no. 23, pp. 18–22, 1996.

[27] F. Corradini, A. Polini, B. Re, and F. Tiezzi, "An operational semantics of BPMN collaboration," in *12th International Conference Formal Aspects of Component Software*, ser. LNCS, vol. 9539. Springer, 2015, pp. 161–180.

[28] J. Meseguer, "Conditional rewriting logic as a unified model of concurrency," *Theoretical computer science*, vol. 96, no. 1, pp. 73–155, 1992.

[29] ——, "Twenty years of rewriting logic," *J. Log. Algebr. Program.*, vol. 81, no. 7-8, pp. 721–781, 2012.

[30] C. Ellison and G. Rosu, "An executable formal semantics of C with applications," in *39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2012, pp. 533–544.

[31] C. Hathhorn, C. Ellison, and G. Rosu, "Defining the undefinedness of C," in *36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015, pp. 336–345.

[32] D. Bogdanas and G. Rosu, "K-java: A complete semantics of java," in *42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 445–456.

[33] D. Park, A. Stefanescu, and G. Rosu, "KJS: a complete formal semantics of javascript," in *36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015, pp. 346–356.

[34] S. Eker, J. Meseguer, and A. Sridharanarayanan, "The Maude LTL model checker," *Electronic Notes in Theoretical Computer Science*, vol. 71, pp. 162–187, 2004.

[35] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[36] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin, "BProVe: Tool Support for Business Process Verification," in *32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017.

[37] M. Kunze, A. Luebbe, M. Weidlich, and M. Weske, "Towards understanding process modeling the case of the BPM academic initiative," in *Business Process Model and Notation*, ser. LNBIP. Springer, 2011, vol. 95, pp. 44–58.

[38] H. Groefsema and D. Bucur, "A survey of formal business process verification: From soundness to variability," in *International Symposium on Business Modeling and Software Design*, 2013, pp. 198–203.

[39] M. Fellman and A. Zasada, "State-of-the-Art of Business Process Compliance Approaches: A Survey," in *7th International Workshop on Enterprise Modeling and Information Systems Architectures*, 2016, pp. 60–63.

[40] P. Van Gorp and R. Dijkman, "A visual token-based formalization of BPMN 2.0 based on in-place transformations," *Information and Software Technology*, vol. 55, no. 2, pp. 365–394, 2013.

[41] P. M. Kwantes, P. Van Gorp, J. Kleijn, and A. Rensink, "Towards Compliance Verification Between Global and Local Process Models," in *Graph Transformation*, ser. LNCS. Springer, 2015, vol. 9151, pp. 221–236.

[42] N. El-Saber and A. Boronat, "BPMN Formalization and Verification Using Maude," in *Workshop on Behaviour Modelling-Foundations and Applications*. ACM, 2014, pp. 1–12.

[43] E. Börger and B. Thalheim, "A Method for Verifiable and Validatable Business Process Modeling," in *Advances in Software Engineering*, ser. LNCS. Springer, 2008, vol. 5316, pp. 59–115.

[44] F. Kossak, C. Illibauer, V. Geist, J. Kubovy, C. Natschlager, T. Ziebermayr, T. Kopetzky, B. Freudenthaler, and K. D. Schewe, *A Rigorous Semantics for BPMN 2.0 Process Diagrams*. Springer, 2014.

[45] W. Huai, X. Liu, and H. Sun, "Towards Trustworthy Composite Service Through Business Process Model Verification," in *7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*. IEEE, 2010, pp. 422–427.

[46] R. Koniewski, A. Dzielinski, and K. Amborski, "Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation

of Multimodal Logistics Chains," in *20th European Conference on Modeling and Simulation*, 2006, pp. 28–31.
[47] M. Ramadan, H. G. Elmongui, and R. Hassan, "BPMN formalisation using coloured petri nets," in *International Conference on Software Engineering & Applications*. ACTA, 2011.
[48] A. Awad, G. Decker, and N. Lohmann, "Diagnosing and Repairing Data Anomalies in Process Models," in *Business Process Management Workshops*, ser. LNBIP. Springer, 2010, vol. 43, pp. 5–16.
[49] J. Ye and W. Song, "Transformation of BPMN Diagrams to YAWL Nets," *Journal of Software*, vol. 5, no. 4, 2010.
[50] G. Decker, R. Dijkman, M. Dumas, and L. García-Bañuelos, "Transforming BPMN diagrams into YAWL nets," in *Business Process Management*, ser. LNCS. Springer, 2008, vol. 5240, pp. 386–389.
[51] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach," in *Business Process Management*, ser. LNCS. Springer, 2015, vol. 9253, pp. 55–71.
[52] ——, "Formal verification of complex business processes based on high-level Petri nets," *Information Sciences*, vol. 385-386, pp. 39–54, 2017.
[53] P. Y. H. Wong and J. Gibbons, "A Process Semantics for BPMN," in *Formal Methods and Software Engineering*, ser. LNCS. Springer, 2008, vol. 5256, pp. 355–374.
[54] F. Puhlmann and M. Weske, "Investigations on Soundness Regarding Lazy Activities," in *Business Process Management*, ser. LNCS. Springer, 2006, vol. 4102, pp. 145–160.
[55] F. Corradini, A. Polini, A. Polzonetti, and B. Re, "Business Processes Verification for e-Government Service Delivery," *Information Systems Management*, vol. 27, no. 4, pp. 293–308, Oct. 2010.
[56] A. Polini, A. Polzonetti, and B. Re, "Formal methods to improve public administration business processes," *RAIRO - Theor. Inf. and Applic.*, vol. 46, no. 2, pp. 203–229, 2012.
[57] F. Corradini, A. Polzonetti, B. Re, and D. Falcioni, "An eclipse plug-in for formal verification of BPMN processes," in *3rd International Conference on Communication Theory, Reliability, and Quality of Service*, June 2010, pp. 144–149.
[58] P. Y. H. Wong and J. Gibbons, "Verifying Business Process Compatibility (Short Paper)," in *8th International Conference on Quality Software*. IEEE, 2008, pp. 126–131.